ESD-TR-71-105

MTR-2050

THE APPLICATION OF MICROPROGRAMMING TECHNOLOGY

J. A. Clapp

MAY 1971

Prepared for

DEPUTY FOR COMMAND & MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 6710
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
F19(628)-71-C-0002

ESD-TR-71-105

MTR-2050

THE APPLICATION OF MICROPROGRAMMING TECHNOLOGY

J. A. Clapp

MAY 1971

Prepared for

DEPUTY FOR COMMAND & MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 6710
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
F19(628)-71-C-0002

FOREWORD

This work was conducted in support of Project 5550 by the MITRE Corporation, Bedford, Mass., under Contract F19(628)-71-C-0002 and was monitored for the U. S. Air Force by Dr. John B. Goodenough, ESD/MCDS.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

JOHN B. GOODENOUGH
Project Officer
Systems Analysis Division

EDMUND P. GAINES, JR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

This report surveys promising applications of microprogramming. Emphasis is on the value of microprogramming as a tool which permits computer users to modify the architecture of a general-purpose machine to better match a particular set of requirements. Factors are discussed which affect the choice of microprogramming over hardware and software in the design and implementation of computer-based systems. Actual and potential examples of its application are given to illustrate its relevance to the solution of implementation and performance problems arising in typical Air Force systems. Finally, research and development tasks are proposed which lead to the realization of the benefits of this technology in operational command control and communications systems. Methods are described for integrating the results into existing and future systems in the next several years.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF TABLES

SECTION I

AN INTRODUCTION TO MICROPROGRAMMING

A DEFINITION OF MICROPROGRAMMING

A computer hardware designer and a computer user, or programmer, have two very distinct views of what a computer is like. To the hardware designer the computer consists of a collection of hardware elements, each of which performs some function, and a set of connections which allow data to flow among these elements. He sees the entire operation of the computer as under the direction of a control unit which can send signals causing data to flow in some predetermined order and in some specified direction within and among the other units of the machine. The programmer who uses the machine, on the other hand, sees himself as entirely directing the operation of the computer through sequences of instructions which make up his software program, whether he programs in a high-level language such as FORTRAN or uses machine language. The program must be translated to machine instructions, each of which is a request to the control unit to activate some sequence of basic hardware functions which actually accomplish the more complex machine instruction, so that both views are true.

Despite the analogy between the way the machine executes a software program and the way the control unit causes a sequence of suboperations to be performed, the methodologies for specifying and implementing hardware logic and software logic were quite different in the computers of the 1950's. The programmer specified a sequence of machine instructions which were stored in the machine memory, and the control unit would step through them, performing the specified operations. The hardware designer conventionally used sequential control logic to specify, for each machine instruction, the block diagram showing all the necessary connections. The control unit consisted of a proliferation of different kinds of hardware and specific fixed, hard-wired logic sequences with many cross-connections for control. The result was so complex that the unit became difficult to understand, maintain, and modify. In 1951 Wilkes [1] proposed a solution to the problem of improving hardware reliability. He showed that by rearranging the structure of the control unit, it would be possible to specify a machine instruction as a sequence of subcommands. The control unit could be viewed as a matrix consisting of one vertical line for each control gate and a set of horizontal lines having access to all possible control gates at the intersections. A machine instruction could then be specified as a set of horizontal lines, each of which was connected to the specific combination of gates it wished to pulse to

control the paths over which data would flow. Thus, the horizontal lines corresponded to subcommands or steps which he termed "micro-operations," and the sequence of steps for one machine instruction was called a "microprogramme." This approach meant that the logic designer, like the software programmer, could specify the sequence of operations to perform some function and that sequence could be stored in the machine. There were now two similar levels of control: the machine instruction of the programmer and the microoperation, or microcode, of the logic designer.

While Wilkes' scheme was elegantly simple, it was not practical at the time because it was extravagantly expensive. Wilkes and his colleagues set about refining the original design to provide greater flexibility, particularly in control of sequencing through the micro-operations. Others sought to improve the timing and reduce the number of control lines. When hardware technology produced faster components, the goal of realizing efficient, economical hardware organizations according to Wilkes' principles was finally reached. Such machines were called "stored logic" computers to emphasize that the logic was not fixed, hard-wire. The place where the microprograms reside is called the control store.

Although the first intention of microprogramming was to make the design of control units more systematic, its implications were more far-reaching. As Wilkes himself observed, the underlying structure of the control unit was now independent of the actual machine instruction set which was implemented. It became possible to select an instruction set much later in the development of the hardware by changing the contents of the control store instead of redesigning and rewiring the control unit. He noted that when the contents of control stores could be replaced, then a programmer would be able to "choose his order code to suit his own requirements and to change it during the course of the programme ...." Machines are available which can permit this freedom to the programmer. It is now incumbent on him to take advantage of it.

There are many contexts in which the term "microprogramming" is used. In this report it designates the <u>process</u> of specifying a machine instruction set as sequences of microinstructions which can be stored in the control store of a computer. The microprograms which are produced are referred to as "firmware," a word coined by Opler [2] to distinguish it from hardware and software.

STATUS OF MICROPROGRAMMING

A brief summary of the current status of microprogramming applications is presented here. For a more complete description, a tutorial paper by Rosin [3], an annotated bibliography [4], and a book by Husson [5] serve as good sources.

The first major application of microprogramming was in the development by IBM of the System 360 series of computers. Each computer type in the series has a different hardware configuration, and some computers include a microprogrammable control store. Microprograms were developed to provide compatibility with earlier IBM equipment and among the computers in the series. The first type of compatibility, called emulation, allows the instruction set of a computer such as the 1410, which is architecturally quite different from a 360 machine, to be interpreted through microcode on the 360. The advantage to a user is that software for one machine can be made to operate on another machine with little or no modification. The success of the emulation technique is dependent on the similarity between the two machines, and under some circumstances the emulated software may run more slowly although the new machine is actually faster. Among the System 360 computers, where most of basic machine architecture is intentionally similar, microprogramming has been used to provide a common instruction set on all machines so that a user can move software from one machine to another. Clearly, such applications of microprogramming benefit the manufacturer in selling machines which are downwards or upwards compatible at the software level. Many other manufacturers have joined IBM in using microprogramming as a means of implementing a machine. Table I shows some current microprogrammable machines. The table is not intended to cover all of the available computers, but rather indicates the broad range of computers for which microprogramming has been deemed appropriate.

Engineers have been prolific in applying new hardware technology such as Large Scale Integration (LSI) to the design of control stores and in developing different kinds of micromachines. Originally microprograms were stored in read-only memories to gain the necessary speed improvement over main memory. There has been a marked trend toward the use of writable control store, sometimes in conjunction with read-only memory. In the recently announced IBM 370/165, read-only memory contains microprograms for control of basic functions and writable control store contains microprograms for control of different modes of machine operation such as emulation and microdiagnosis. On current computers there are many different implementations of microprogram storage, some of which can only be altered by physical replacement; others are mechanically or electronically alterable. The ease of

Table I

A Survey of Microprogrammable Computers*

	DATE	MAIN MEMORY WORD SIZE (BITS)	MAIN MEMORY RANGE (1024 WORDS)	MAIN MEMORY DATA PATH (BITS)	MAIN MEMORY CYCLE TIME (μSECS)	CONTROL STORE TYPE	CONTROL STORE READ ONLY OR WRITABLE	CONTROL STORE WORD LENGTH (BITS)	CONTROL STORE CAPACITY (WORDS)	PROCESSOR CYCLE TIME (NANOSECONDS)
Cincinnati Milacron										
CIP-2000	1969	8	4-32	8	1.1	Diode	R	16	1024	220
CIP-4000	1970	18	4-64	18	1.2	IC Chips	R	80	1024	400-1000
Digital Scientific										
Meta-4	1970	16	8-64	16	.9	Air Coupled Induction	R	32	2048	85
IBM										
360/25	1967	8	16-48	16	1.8	CORE	W (1)	16	8192	900
360/30	1965	8	8-64	8	1.5	Card Capacitor	R	60	4096	750
360/40	1965	8	8-256	16	2.5	Transformer	R	60	4096	625
360/50	1965	8	32-512	32	2.0	Balanced Capacitor	R	90	2816	500
360/65	1966	8	128-1024	64	.75	Balanced Capacitor	R	100	2816	125
360/85	1969	8	512-4096	128	1.0(3)	Monolithic	R,W	128	2500	80
370/145	1970	8	112-512	64	.540/.608(2)	Monolithic	W (1)	32	16384	202.5-315.0
370/155	1970	8	256-2048	128	2.07(3)	Monolithic	R	72	8192	115
370/165	1970	8	512-3072	64	2.0(3)	Monolithic	R,W	108	2560	80
Interdata										
3	1967	8	4-64	16	.98	Transformer	R	16	4096	370
4	1968	8	4-64	16	.98	Transformer	R	16	3500	370
5	1970	8	8-64	16	.98	Transformer	R	16	1700	370
Micro Systems										
Micro 800	1969	8	4-32	8	1.1	Diode	R	16	1024	220
Raytheon										
RAC 251	1970	32	4-64	32	1.8	Bipolar LSI	R	96	512	350
RCA										
Spectra 70/45	1966	8	16-256	16	1.44	Transformer	R	54	2048	480
Spiras										
65	1969	16	4-64	16	1.8	Braided Wire	R	32	1024	450
Standard Computer Corp.										
MLP-900	1970	36	32-4096	36	.7	Semiconductor	W	32	4096	128

Notes

1. Main storage and control storage reside in the same physical storage.
2. Fetch/store times.
3. Also equipped with an auxiliary high speed buffer storage which reduces the effective cycle time.

*Prepared by R. H. Bullen

changing microprograms, the amount of architectural variation possible through microprogramming, and the cost depend on the specific machine. However, control stores tend to be relatively expensive and hence much smaller than other kinds of storage in today's computers; a judicious use must be made of the available space.

Manufacturers have provided their own impetus for the development and application of microprogrammable machines. Users have not been as actively involved in modifying the architecture of delivered machines to tailor them to their own requirements for machine instruction sets and computer organization. There are several reasons for this:

1. Many manufacturers have been reluctant to allow customers to do their own microprogramming, primarily because they do not wish to maintain delivered hardware and software under varying machine architectures which might lead to incompatibilities. IBM, in particular, has resisted the release of microprogramming capabilities to users, although their machines have been modified to customer specifications on an RPQ basis. Other companies, such as Burroughs, provide engineering support or allow microprogramming with administrative constraints.

2. Microprogramming, while similar to software programming, requires more detailed knowledge of machine hardware characteristics; and timing must be considered in selecting sequences of microinstructions. In general, the micromachine is not as easy to deal with, and the support tools for coding and checking out microprograms have not been as sophisticated as those available for implementing software.

3. Users do not fully understand the benefits to be gained from microprogramming and when it is appropriate to apply it, although the idea of microprogramming has been around for twenty years. The newness of interest in this technology is best demonstrated by the fact that the first book on the subject was published in 1970 [5]. The number of documented, proven applications of microprogramming to enhance performance for particular kinds of computer processing is accelerating, which would indicate that users are learning of its value.

THE FUTURE

Most attempts to predict the important computer hardware and software developments over the next decade include forecasts indicating the emerging importance of microprogramming. In 1967 Opler [2] predicted that microprogramming, or firmware as he called it, would

assume a dominant role in structuring a computer because present software is too complex, too expensive and too slow. A trade off between software and firmware would obtain the greatest price/performance improvement. Amdahl [6] felt that microprogramming would be important in the '70's because its proven effectiveness as a design technique will make it more widely available on machines, especially since LSI technology makes it possible at a reasonable cost. Withington [7], who forecast technological trends likely to affect Management Information Systems technology over the next five years, stated that such systems will be constrained by software capabilities far more than hardware. Software, according to his studies, must be made more reliable, efficient and convenient to use. He sees microprogramming as a technique for enhancing software efficiency. A survey [8] of computer manufacturers indicates that although they still have reservations about making it available to users, microprogramming has received approval as a technology. To add yet another prediction, it seems clear that manufacturers are introducing small and large computers with the express purpose of allowing customers to microprogram them; it is also clear that the use of minicomputers for dedicated tasks is on the upswing. Microprogramming of minicomputers to support these dedicated tasks will be one of the obvious applications of microprogramming. Emulation is another use of microprogramming which will continue in the future. Much more effort will be put into using microprogramming to make significant changes in machine architecture to support some of the difficult problems with software existing today, in particular its performance and reliability. Even after new hardware becomes available in a practical sense to allow new kinds of machine and memory organization, microprogramming will continue to be important as a means of creating the machine instructions in the logic unit which make the hardware accessible to the programmer. The most immediate problem in the application of microprogramming by computer users is to determine when and for what it is appropriate.

SECTION II

DESIGN CONSIDERATIONS IN THE APPLICATION OF MICROPROGRAMMING

INTRODUCTION

One of the reasons, given earlier, for the delay in users' acceptance of microprogramming was a lack of understanding of its suitability to their problems. General principles for determining when it is appropriate to use microprogramming have not been available to users. It is clear that what may be an advantage in one set of circumstances may be a disadvantage in another. Nevertheless, there are common processes used in determining the suitability of any tool, including microprogramming, to a particular problem. Undoubtedly the same factors will be involved even when each user gives his own weight to those factors in reaching a final decision. It is therefore worthwhile to develop these principles to clarify the choices presented by microprogramming to systems designers. In this section, some of the benefits, as well as the limitations of microprogramming, are discussed in relation to other alternatives available to a designer of a computer-based system. The rationale for design choices is the effect on design as well as on the implementation or fabrication phase and on the system performance and cost. One problem which faces the designer is how to make use of the resources available to him to accomplish the functions needed to meet the system's requirements. Conventionally, the major resources for obtaining functional capabilities have been hardware and software, until the advent of microprogramming which now offers a third alternative. It is the responsibility of the designer to consider the factors involved and to analyze the effects of distributing processing among these resources. The final decision is very complex because of the interaction among the relevant alternatives, which necessitate trade offs among the factors which contribute to a particular set of requirements. In order to simplify the discussion of major factors in the selection of microprogramming as a resource, it will be compared first with hardware and then with software, with each factor considered independently of the others.

DEFINITIONS

Each of the three resources being compared is defined in terms of its use to accomplish some complex functional requirement of a system. The term "hardware" refers to the use of hard-wired equipment which is not modified after its delivery; e.g., a controller for a particular kind of peripheral device or the physical equipment of a

computer. "Microprogramming," as a resource, refers to the firmware which modifies the behavior of a computer to make it perform some function. "Software" implies the use of computer programs to perform the required function.

MICROPROGRAMMING VERSUS HARDWARE

Flexibility

Microprogramming offers the flexibility to modify general-purpose hardware without making hardware modifications. Flexibility is probably the most important advantage of firmware over hardware. This kind of flexibility can be useful in the following ways:

1. To meet unanticipated kinds of requirements.

   A typical application of this kind of flexibility would be the handling of peripheral devices, whether by a separate peripheral processor or by a general-purpose computer. If a new kind of device is introduced, hardware cannot be easily adapted to handle device characteristics for which it was not originally designed. If the device handling is performed in a microprogram, it can be recoded to allow for the new device characteristics.

2. To upgrade existing capabilities.

   If a processing function is microprogrammed, then it is possible to upgrade that function's capabilities or performance as new techniques become available. For example, on the 360/50, an improvement in the accuracy of floating point arithmetic was made by rewriting the microcode. This change was easily installed in computers in the field by a replacement of the microprogram.

3. To standardize procurements of hardware.

   A single type of general-purpose, microprogrammable computer can be procured for use in performing different functions in different parts of a system or even in different applications or environments. Microprogramming can be used to adapt each of the computers for those specialized functions it must perform. The Logicon 2 + 2 system consists of an integrated set of four minicomputers, three of which are identical models with different microprograms. One is a control processor, one is a peripheral processor, and the third is an application processor [9].

4. To use a computer for more than one purpose.

Through microprogramming it is possible to change the mode of operation of a single processor to serve more than one purpose or to combine functions which would normally be performed by different special-purpose hardware processors. The IBM 360/40 has two modes of operation: it can be used for normal 360 instructions or to emulate the machine instructions of a 1410 computer. The change from one mode to another can be accomplished in different ways depending on the hardware configuration; e.g., the microprogram on an Interdata 3 must be changed by physically replacing read-only memories; the IBM 370/145 has a writable control store which can be loaded from a special disk. Some control stores may be modified dynamically as easily as core memory and, in fact, may reside there.

An example of a single microprogram to perform the functions of many hardware devices is an integrated communications controller [10], which reduced the need for unique hardware channels and control units for a wide variety of communications devices by handling the functions for these devices in one microprogram.

Cost

If the alternatives in achieving a complex functional capability are special-purpose hardware or microprogramming a general-purpose computer, then microprogramming can often reduce the time and cost of custom design and fabrication and can result in smaller maintenance costs. LSI technology is making the microprogrammed approach more economically feasible. In one comparison of a microprogrammed aerospace computer and a similar non-microprogrammed computer [11], LSI technology was not used; and the microprogrammed machine was more expensive by about the cost of the read-only memory. However, it was estimated that about 85% of the logic modules external to the microprogram storage were amenable to LSI implementation compared with about 50% for the non-microprogrammed machine. Predicated on a lower gate cost for LSI than unit logic, the microprogrammed computer would be cheaper. Another report [12] substantiates the potentially lower cost of microprogrammed machines with LSI circuits. The inherent flexibility of microprogrammed machines can also reduce the cost of changes. New capabilities can usually be added to a microprogrammed machine with no additional hardware cost, except when additional control memory is needed, whereas additional logic in a special-purpose machine requires additional hardware.

There is a cost savings which may be realized by replacing several hardware units with one processor microprogrammed to perform the combined functions of the other units. Some hardware duplication, such as separate power supplies, can be avoided. Functions which were the same in different units can be implemented just once.

Speed

The execution time for a function which is hard-wired will often be shorter than for the equivalent function executed in microcode, assuming both methods are well-designed. Hard-wired functions in computers can usually be made to operate at memory speed. Whether the same is true for microcode will, in part, depend on the ratio of microinstruction cycle time to memory speed; i.e., how many microinstructions can be executed in one memory cycle. A hard-wired approach allows more operations to occur concurrently than microcode permits; e.g., a multiply operation can be done more quickly in hardware through a set of parallel shifts and adds, while a sequence of microinstructions is needed. Every machine instruction which is executed must be decoded. In a microprogrammed machine, the microinstructions must also be decoded, although this may not be significant in the overall execution time.

Reliability

A microprogrammed computer has a simpler, more regular design for its control logic with fewer different kinds of hardware components than a hard-wired or sequential logic control unit. Wilkes' [1] original motive for microprogramming was to improve the reliability of control units of computers, and it was the driving force behind manufacturer acceptance of this approach to logic design. When comparing the use of microprogramming to special-purpose hardware for accomplishing a particular function, the advantage of microprogramming increases with the complexity of the function. There are simple functions which might result in simple hardware logic which, in turn, might be more reliable because of its simplicity.

When the same set of hardware can be microprogrammed to serve different functions, a facility may contain fewer machines or several identical machines dedicated to special processes which can be maintained in similar ways. Alternatively, a set of special-purpose processors which are physically different might each require a different set of maintenance procedures. Reliability should improve when maintenance is simplified. Identical machines performing different functions through microprogrammed specialization offer the advantage of being interchangeable at the hardware level. The implication is that the total reliability of a system configuration can be improved when any of the identical machines is a potential replacement for another one

which has failed. Even if it has been performing a different function, a machine's microprogram can be changed to allow it to assume the role of a machine which is down. For example, if a configuration contains processors and I/O controllers which are all using the same kind of machine with different microprograms, then it is possible to configure them so that an I/O controller can become a processor if a processor fails and vice versa, depending on which function is considered more important to sustain.

One of the significant contributions that microprogramming can make to improve the reliability of operational systems is to dynamically perform error checking and correction and to achieve fail-soft operation by dynamically changing algorithms to bypass hardware components which have failed. This role for microprogramming is discussed in greater detail below in the comparison with software, where the trade offs can be identified more easily. Specific requirements must be known to determine the effectiveness of microprogramming in contrast to hardware circuitry for the detection and correction of hardware failures. Microprogramming can be used in conjunction with hardware to provide slower, but functionally equivalent, capability to be used in the event of hardware failure. In the IBM 360 model 85, there is a low speed multiply algorithm in microcode which can be used when there is a failure or malfunction in the high speed multiply hardware [13].

MICROPROGRAMMING VERSUS SOFTWARE

Speed

The major advantage that has been promulgated for the use of microprogramming to replace software has been its greater speed, which can enhance the performance of systems. It is not an a priori truth that a software function can be replaced by a microprogram, and its execution time will be shorter. The actual performance ratio is a function of the sequence of operations which must be performed as well as the performance characteristics of the microinstruction set. The kind of software function which is most suited to replacement by a microprogram has one or more of the following attributes:

1. It is CPU-bound, rather than dependent on input/output transfer time.

2. It produces many intermediate results which are used in the processing and do not have to be preserved when the process is complete.

3. It is highly repetitive, either internally within the process or because it is frequently used as a part of some larger process, and consumes a significant portion of total execution time.

4. It is awkward to do with the existing instruction set and more natural to the microinstruction set.

5. The machine instructions used to perform the function have a high ratio of overhead time, spent on instruction fetch and address generation, to actual operation execution time.

Typically, logical functions which must be repeated many times are good candidates for microprogramming. Complex computational functions which use machine instructions which are inherently slow may not yield as large a performance improvement by replacing software with firmware.

The characteristics of microprograms which affect performance are the following:

1. The speed of execution of microinstructions and its ratio to main memory speed. Where memory accesses for data are not involved, this ratio roughly indicates the number of microinstructions that can be executed in the time of one machine instruction, since each machine instruction must be fetched from memory to be executed.

2. The amount of parallelism in microinstructions. Often a single microinstruction can cause several operations to occur simultaneously, whereas the machine instruction set may not.

3. The width and scope of the data paths for operands of microinstructions. Where the size of operand at the machine instruction level may be fixed for convenience in defining a general-purpose instruction set, the hardware may support different sizes of operands; and the microinstruction set may have more complete access to processor registers, which makes it better suited to some processes.

When the proper match exists between the process and the microprogramming facilities of the machine, then efficiency gains may accrue for any of the following reasons:

1. A single instruction performs the function of many software instructions which must be fetched and decoded. The collateral saving in eliminating core storage space for the software instruction sequence can increase the utilization of core memory and

also result in a performance improvement when many copies of the same sequence would otherwise occupy core memory. Subroutines, which are the software equivalent of saving memory space by avoiding multiple copies of code, exact an execution time penalty for linkage to and from the subroutine.

2. The number of memory accesses for fetching and storing data is reduced. In this case, the data may be constants which are stored in the microprogram or intermediate results stored in faster hardware storage accessible to the microinstructions; e.g., registers or scratchpad memory.

3. The nature of the microinstructions makes possible more efficient algorithms. As an example, a square root function was coded in software, using the Newton-Raphson method; and the equivalent function was microprogrammed using an algorithm which computed a bit at a time instead of using shift and divide operations (14). While the results are dependent on main storage time, the example showed the way in which time was spent in each case; and the microprogrammed version was eight times faster for 16-bit numbers. In another study of a hypothetical machine with writable microstorage (15), a completely microprogrammed square root program was only 20 percent faster than the equivalent version in IBM 360 Assembly language. This contrast in results serves to underscore the fact that the efficacy of microprogramming as an alternative to software is dependent on the hardware resources for writing microprograms as well as the algorithms chosen for applying them.

Before the system designer can fruitfully apply microprogramming to gain efficiency, he must determine what parts of a process might be suitable for microprogramming. For a system which is not operational, such an analysis might require a simulation of the operating times. Next, the processes to be optimized must be analyzed to see if the microprogramming facilities available can be of any use in performing the same functions faster, in the same or different ways, than software. The generalizations above can be tested to see which apply to the particular situation.

Cost

There are two distinct ways in which the trade off between software and microprogramming can be viewed as affecting the cost of developing a system: the use of microprogramming to lower software costs and the comparative costs of producing software and microprograms. It is well known that the cost of procuring a system is primarily the

cost of procuring the software rather than the hardware, with estimates as high as 70 percent attributable to software [16]. While some of these costs can be mitigated by performing more software functions in hardware and firmware, it is an oversimplification if the cost of the hardware and microprograms are not also considered. It should be obvious that the implementation of microprograms is itself a task not unlike that of producing software. It requires a design phase, a coding phase, and a checkout phase. There is a parallel between the early stages of the development of software technology, when time and attention were given to programming languages and tools, and the current interest in the same problems for microprogrammers. Presently, the aids to microprogramming have not yet reached the level of sophistication of software techniques, although microprogrammers will undoubtedly borrow heavily from their software counterparts.

There are significant differences between microprogramming and software programming which make microprogramming a more difficult process [17]. The microprogrammer has to understand much more about the hardware, including a consideration of timing between instructions. Microinstructions on some machines support parallel execution of operations by one instruction, which is more difficult to contend with than sequential operation. The microprogrammer is often dependent on a simulator to check out his microprograms, particularly if they will eventually reside in read-only memory. Simulators cannot always reproduce timing and detect timing errors; therefore, complete checkout is not possible. It has also been noted [18] that the slowness of some simulated checkout tends to discourage completeness in testing. Since the cost of control stores and working storage for microprograms are usually expensive, and they are therefore small, difficulties in producing microprograms can arise from the complexity introduced by limitations in program and data space. A further drawback to microprogramming arises if it is difficult or expensive to modify a microprogram after it has been installed; namely, corrections and additions may have to be made by patching in a way that minimizes the number of physical changes but obscures the logic. The trend is toward writable control stores and techniques which should lower the cost of modifying microprograms in the newer machines and toward better languages and support tools for writing and testing microprograms.

A more subtle use of microprogramming to reduce the cost of software production is to change the architecture of the machine, and its instruction set, to more closely match the way in which the programmer of software would like to use the machine. The programming task can then be simplified, which would not only make programmers more productive, but also reduce the number of coding errors. One experiment in microprogramming has been performed [18] with this objective in mind. Functions such as storage and process management, which interfere with

or distract from the logical requirements of a system, were removed from the purview of the programmer and placed in the microprogram. There are architectural changes which can be made to facilitate the checking out of software as well. By making it easier to monitor the behavior of software while it is being executed, the cost incurred by debugging time can be reduced.

Another facet of the use of microprogramming to lower the cost of producing software is presented when there is a large investment in software for a particular machine, and it is necessary to replace the machine. In order to preserve the software, it may be more economical to make the new hardware behave like the old hardware at the software level, by emulating the instruction set of the machine for which the software was written. Many factors influence the effectiveness of this approach [19]. In most cases, the new machine is faster than the old machine. While emulation may produce the equivalent of faster execution time for the emulated software, even greater performance improvements might be achieved by redesigning and recoding the software to take advantage of the new machine architecture. If the two machines are sufficiently different, then the ways of processing that were originally defined may no longer be appropriate. Emulation will be discussed further in Section IV.

Reliability

Physically, microprograms tend to be more reliable than software because they are not subject to modification or destruction. Read-only memories cannot be modified by the execution of firmware or software. Many writable control stores can only be loaded with microprograms which cannot modify themselves during operation, and some writable control stores are only written under control of instructions in the read-only memory, as in the IBM 360/85.

Firmware can be more effective than software in maintaining the reliability of machine hardware. Microdiagnostics have been successfully used on several IBM machines [13, 20], including the 370 series, instead of software. Microinstructions require fewer hardware circuits and can therefore be executed using a smaller hard core, the part of the machine required to be working before diagnostics can be performed. Microinstructions deal more specifically with hardware components than software and can disable parts of the system to localize errors more precisely. In the 360/30, it was reported that microdiagnostics reduced the hard core from 50 percent to 1 percent of the CPU and allowed location of more than 90 percent of the failures in the CPU with very high resolution [20]. In cases where parts of the hardware are not accessible through microinstructions or the control store is too small for complete diagnostics programs, a combination of firmware and software

is feasible. The application of microdiagnostics during system execution, as mentioned earlier, can result in fail-soft operation by detecting hardware failures and substituting alternate means of achieving the same functions.

Microprogramming can also be used to improve the reliability of software by causing architectural changes which simplify the programming process or standardize the way in which important operations are accomplished. Hopefully the programmer is then less prone to make coding errors.

Protection

When it is physically impossible to read the contents of a control memory with software, then vital algorithms and data contained there cannot be compromised. Whereas rules can be imposed which proscribe performing certain functions in software, microprograms can be written which actually prevent undesirable operations from being performed; e.g., a user can be prevented from accessing or modifying the contents of parts of memory which do not belong to him. This kind of control is necessary to protect system software from users and, in multiprogramming, to prevent independent users from interfering with each other. At least one computer, the Gemini under development by Computer Operations Inc., has special architectural features provided by firmware and software for the protection of data from unauthorized access [21].

Capability

Microprogramming makes it possible to add capabilities which are not achievable, in a practical sense, in software. A simple but useful example is a bootstrap loader. In many machines a method is needed to initiate the loading of software into a "bare" machine. A program to accomplish this is called a bootstrap loader, since it usually contains the minimum amount of code necessary to start reading from a peripheral device containing a more sophisticated program to load the remainder of a system. On small machines, it is not unusual to find that the bootstrap loader must be manually entered through switches into core memory. In cases where it is wired in, the loader will only start a specific device. A bootstrap function can easily be accomplished through a microprogram which is activated by some manual control [22]. At MITRE, such a bootstrap was microprogrammed to load either punched paper tape or cards, depending on a switch setting [23].

Another kind of capability made possible through microprogramming is the ability to make a function indivisible. This can be explained by considering the effect of interrupts on the execution of some critical operation involving several instructions. An interrupt can be

defined as the suspension of execution of a sequence of code at the completion of any instruction in the sequence in order to perform another sequence which may or may not be related. This kind of interrupt occurs in multiprogramming environments when the processor is switched from one task to another. There are critical sequences which should not be interrupted until they are complete, or the integrity of the function they perform may be destroyed. This is the case if the code executed changes values being used by the interrupted function. By making such a function a single machine instruction and performing the sequence of operations in a microprogram rather than software, the operation can be made indivisible with respect to interruption.

CONCLUSIONS

Microprogramming, under the right circumstances, is a viable alternative to hardware or software for accomplishing a function, whether because it is easier, more efficient, more economical or more flexible. The primary advantage of microprogramming a general-purpose computer over employing specialized hardware is the flexibility of the microprogrammed machine. On a cost basis, the microprogrammed machine is becoming competitive, while on a reliability basis, it may surpass its "hard-wired" equivalent.

The primary advantages of microprogramming with respect to software are collaborative, supporting the software where greater performance or capability is needed and modifying the machine architecture to make software production a simpler process.

SECTION III

EXAMPLES OF THE APPLICATION OF MICROPROGRAMMING

INTRODUCTION

The previous section described the major factors involved in deciding when to use microprogramming. Some actual examples of its use may better illustrate how microprogramming can be integrated with hardware and software to yield a total solution to the problems of specific application areas. The magnitude of the effect of using firmware will be indicated wherever possible.

INPUT/OUTPUT PROCESSING

An area of application of microprogramming which is developing rapidly is input/output processing, characterized by communications processing and control of peripheral devices. Some of the salient features of this kind of processing can be summarized as follows:

1. It is device-specific, dealing with the characteristics of individual devices such as character codes, formats, and other conventions for addressing devices.

2. Timing is usually important.

3. The processing must be responsive to interrupts because it is initiated by inputs to a System from a device, in contrast to other kinds of processing which are initiated by the processor and can be performed in some sequence or according to some schedule.

4. The processing covers a fairly narrow range of functions which are frequently repeated; e.g., assembling and disassembling characters. These functions can usually be done efficiently in real-time at the channel level because they are simple and have few parameters.

5. Large amounts of high-speed memory are not usually required, except for buffers, in contrast to other data processing where programs and data for a process may be large.

Examples of input/output processing can include polling, multiplexing, routing and addressing messages, generating transmission codes, error detection and correction, data compression, as well as more sophisticated processes such as editing and source data automation.

The methods of performing input/output functions range from exclusive use of hardware, as in device controllers or communications equipment, to combinations of hardware, firmware, and software. With any of these methods, the processing can be performed within the mainframe processor or in a separate processor. It is claimed [24] that a large-scale computer can be impaired by up to 40 percent of its throughput capacity if it does its own data communications housekeeping. There are other cogent arguments to support the use of separate minicomputers for input/output processing: the software of the large machine is simpler when device-specific code is removed, and device interrupts need not be handled; the overhead operations in I/O processing and the space for code to perform these operations can be eliminated to make more efficient use of the computer. The shorter word length and smaller high-speed storage of a minicomputer do not preclude performing I/O processing. When the I/O processor is programmable, then it has the flexibility to add new functions and to change processing methods to provide new capabilities or to handle new devices. The characteristics of I/O processing described above are all compatible with the capabilities of firmware described in the previous section. The use of microprogramming for I/O processing enhances the processor performance because microprogramming is closer to the hardware and can often be used to control devices more easily, more efficiently and more closely than software.

There are numerous examples of microprogrammed input/output processing which have been described in the literature [e.g., [10], [25], [26], [29]]. Manufacturers are also marketing communications processors and device controllers which are microprogrammed and can compete favorably on a cost basis with hard-wired controllers. Control of multiple devices of the same or different kinds are usually accommodated with a single microprogram.

REAL-TIME DATA PROCESSING

Real-time data processing is typified by the processing of data which is collected at frequent intervals and referred to as real-time data. This class can include processing which happens in direct response to the receipt of the data, usually in order to perform some control function; e.g., guidance or navigation control using sensor inputs. Analysis of real-time data to produce reports can also be included in this class. Examples of real-time data are signals, whether from radars,

acoustic or optical devices. Examples of real-time data processing are multi-sensor correlation, spectral analysis, pattern recognition, and filtering of digital signals.

In some ways input/output processing is similar to real-time data processing, but a distinction can be made by emphasizing the following characteristics of real-time data processing:

1. Large volumes of data are involved.

2. Efficiency of processing is important because of the volume of data for which processing must be done. Response time is especially important in control situations. Even when the data is being used for analysis or simulation, the amount of computer time can become exorbitant.

3. The repetitive operations which are performed tend to be computational in nature, as opposed to the bit and character manipulation operations of input/output processing.

4. In many cases, the functions are performed on arrays of data. This implies large amounts of data must be accessed to perform one cycle of processing. It can also imply that parallel execution of operations on more than one set of data at a time is suitable for some of the processes.

Microprogramming can be employed in several ways to effect an improvement in performance for real-time data processing. The extent to which implementation of arithmetic functions in microprogramming rather than software can be beneficial is a function of the nature of the microprogrammed machine architecture and the nature of the algorithms. In an internal IBM study quoted in [5], it was stated that the improvement for microprogrammed composite arithmetic operations is small, on the order of 1.2 to 1.6, because the instruction fetch time saved is such a small proportion of the time spent on performing arithmetic operations. On the other hand, an implementation of a matrix inversion instruction in microprogram [28] for two models of the IBM 360 gave a 3:1 improvement over equivalent software, and a square root routine was eight times faster than the equivalent software [14]. Several special-purpose processors have been implemented in microprogramming to process real-time signals efficiently and perform functions such as Fast Fourier Transforms (e.g., 12). Such processors can be used to distribute processing in a real-time system. The most powerful application of microprogramming for speed up of real-time data processing appears to be in conjunction with special-purpose hardware. One such case is the IBM 2938 Array Processor [29], which can be used with several models of the IBM 360 computer by attaching it as a channel-driven

device using standard input/output commands. The Array Processor can run concurrently with the CPU of the 360 computer to perform a small set of vector and matrix operations. It has a high-speed arithmetic unit capable of performing high-speed floating-point multiply-adds. The logic to accomplish the operations is microprogrammed in the Array Processor. Measurements of performance improvements indicated that the use of the Array Processor allowed a complex operation such as a convolving multiply to be 250 times faster, and a matrix multiply was 40 to 50 times faster. It was noted that the improvement was greater for larger arrays and for more complex operations. In comparing its use with different machines, it was noted that the greatest improvement was on the least powerful machine. Another case, described in [5] also involved signal array processing to be performed every 50 milliseconds. Execution time in software would have required 545 milliseconds. The final approach, which reduced the time to 49 milliseconds, used a combination of microprogramming and a hardware multiplier-summation processor. While microprogramming was only one source of the performance improvement and its exact contribution was not stated, it played a central role in adding special-purpose high-speed hardware to a system without causing perturbations in the behavior of its existing hardware and software.

AIRBORNE SYSTEMS

An airborne system which uses a digital computer has many of the requirements found in input/output processing and real-time data processing. Additional requirements of airborne computer systems are very high reliability, compact size, and minimized weight. The hardware technology of microprogrammed machines, particularly with LSI components, should make them more compact and more reliable than their hard-wired logic counterparts. In previous sections, the use of microprogramming to improve reliability by dynamically performing diagnostics in firmware has been described. The RCA VIC computer [30] was a microprogrammed machine designed with high reliability in mind for airborne applications. Because of its experimental nature, not all of the intended design was realized, but the principles are valid. The hardware emphasized reliability without restricting the flexibility to design new instruction sets. The RCA VIC is being used at SAC and illustrates another application of microprogramming which is relevant to airborne systems; namely, the ability to emulate a portion of the instruction set of the ground computer in the airborne machine. This provides sufficient compatibility to allow software for the airborne computer to be developed on the larger ground computer with the aid of software tools which may not be available on the smaller machine. Compatibility of instruction sets can also allow common software for parts of the systems on each machine, despite disparities in hardware.

There are several examples of microprogrammed airborne or aerospace computers already in existence, which perform real-time computations (11, 30). In examining the trends in digital systems aboard aircraft, a study noted that microprogramming will be emphasized as a means of economizing on storage capacity and processing time (31). An approach to achieving this economy is through the development of machines whose instruction sets can directly execute higher-level language programs such as Jovial or Space Programming Language. Several Avionics laboratories in the Air Force and the Navy are initiating research into the architecture of such machines and the practical means of achieving them. The feasibility of this approach was demonstrated by Weber (32), who produced a microprogrammed compiler and an interpreter for an Algol-like language. The compiler translates statements in the language to an intermediate language which matches the high-order language and, at the same time, can be interpreted by microprogrammed machine instructions. The advantage of this approach to airborne applications is that programs represented in the intermediate language are more concise and therefore require less storage space than their expansion into the usual machine instructions. Gains in speed of execution as high as an order of magnitude over the equivalent software can also be realized for some sequences of operations. Airborne digital systems are assuming data management functions, in addition to navigation, as exemplified by the Advanced Airborne Command Post. As the size of the data base increases, the efficient use of storage becomes more urgent in order to conserve space and weight.

In proposing the design of a computer to meet the Naval airborne data processing requirements for 1975 to 1985, Entner (33) revealed another aspect of microprogramming for airborne applications. His concern was to meet processing requirements and also reduce the enormous cost of computer procurement. His solution is standardized modular building blocks which can have their control structure dynamically reconfigured through microprogramming. An installation can consist of several specialized processors under central control. Current achievements and ongoing developments indicate that the contribution that can be made by microprogramming to airborne digital systems is significant.

SECTION IV

MICROPROGRAMMING RESEARCH AND DEVELOPMENT

RATIONALE FOR RESEARCH AND DEVELOPMENT

There are many ways in which microprogramming is already used profitably, and many areas in which it can clearly be of benefit. Research and development tasks are proposed below which show high probability of payoff in solving some of the urgent problems existing in a broad spectrum of computer-based systems which are of immediate relevance to the Air Force. These systems can be characterized as requiring large amounts of software to do complex processing. Operational requirements demand that they be reliable, and constraints are placed on processing time. Examples of such systems abound in real-time, command control and communications areas. The Advanced Airborne Command Post (AABNCP) and the MAC Integrated Management System (MACIMS) are both representative. The number of such systems being formulated is accelerating at a rate which is not commensurate with our ability to produce them at a reasonable cost and according to the standards expected. The size and complexity of software has an adverse effect on its reliability and response time; conversely, the need for greater capability and faster response time leads to larger and more complex software.

Areas of research and development have been chosen which directly relate to these problems. While microprogramming can speed up specific parts of specific applications by replacing software with faster firmware, this approach can be self-limiting. Such uses of microprogramming tend to be narrow in scope or relevance to other situations, since they represent a careful optimization for a particular set of conditions. It is the intention of the activities below to focus on general problems for a wide range of systems like those in use or contemplated by the Air Force. Microprogramming is a proven tool for tailoring machines to efficiently handle classes of applications. Effective use of microprogramming to solve a particular problem requires that the causes, as well as the results of the problem, be fully understood before constructive steps can be taken. It is therefore a part of these tasks to identify the causes and demonstrate how microprogramming alleviates them.

One of the tasks described below applies microprogramming to improve the performance of Data Management Systems. A second task employs microprogramming to enhance the usefulness of such systems in environments which require processing of data with multiple levels of security classification from remote terminals. A third task addresses the problem which pervades all of the large, complex systems; namely, the need for producing reliable software in a reasonable amount of time. The fourth task is concerned with the control of multiprogramming and multiprocessing, techniques which can improve the overall performance of multi-user systems and the utility of hardware. In particular, the application of microprogramming to operating systems is considered. Finally, it is suggested that improved methods of producing firmware should be supported to help deliver the products which result from the other tasks.

The conduct of such tasks by the Air Force might serve to mitigate some of the criticisms leveled at the Department of Defense by the Blue Ribbon Defense Panel [34]:

> "There is no significant software systems design capability in the Department. Such capability as exists is widely dispersed and focused on narrow spectrums, usually tied to specific applications. As a consequence, no effective mechanism exists for development of more flexible languages, compilers, executive monitors, data storage and retrieval software, operating systems, translators and liberation programs, etc. Current practise makes the Department highly dependent on hardware manufacturers for design of system software. The manufacturers have no incentive to provide increased flexibility to the Department which might increase the Department's independence of the supplier's particular machine and increase Department-wide compatibility of ADP programs."

IMPACT OF RESULTS

The kinds of tasks proposed below will serve to demonstrate, in a laboratory environment, principles which can be applied in operational situations and to prove their usefulness. A period of two years should be sufficient to select, implement, and evaluate techniques for applying microprogramming in the designated areas. At the end of that time period, the recommendations which are made can obviously be of value to planners of systems which have not yet been implemented. For these systems there is still freedom to influence the selection of computers or to define the architecture of an already available microprogrammable computer. The application of these principles is

not limited to systems under development. Microprogramming also permits the integration of changes in machine architecture into existing operational environments. Several methods are available, depending on what kinds of changes or improvements are desired and what parts of the system can, or must, be changed. Typically, hardware configurations may change as new types of peripheral devices become available. Software may have to be changed to meet new functional requirements. Improvements may be necessary because the system is too slow or the capacity too small. A major consideration is to preserve as much of the existing software as possible, because it represents a considerable investment in time and money. Some of the methods for accomplishing this are summarized below.

Adding To An Existing Computer

When any computer in a system is microprogrammable, then it can be modified; yet all of the capabilities and characteristics originally available may be retained, so that existing software continues to operate. The following examples illustrate how this technique can be used:

Adding New Instructions

By adding new instructions to an existing computer, it is possible to redistribute the processing between the software and the firmware. This can be done to achieve greater speed for particular processes or to add new processing capabilities not previously available in the instruction set of the computer. Software can then be selectively modified to take advantage of the new instructions.

Re-implementing Existing Instructions

In this case the instruction set remains the same; but the algorithms for accomplishing instructions might be modified to be more efficient, or new functions might be added that are transparent to the user. The latter approach can be used to accommodate new types of hardware or to add error checking. An example of this occurred when an associative memory was added to a 360/40 computer, and it was necessary to modify the microprogram for all I/O processing without changing its outward behavior [35].

Architectural Changes To The Computer

A computer may be microprogrammed to provide new primitives in the instruction set that are appropriate to some class of applications. New capabilities can be added which are transparent to the instruction set, such as task management on the Venus machine [23]. The addressing

scheme might be modified or the class of operands. In this case, the instructions and their behavior may be considerably different. Such changes may be motivated by a better understanding of the kinds of primitives that are conceptually employed in the application processing and can provide a better mapping of the requirements onto the computing resources. If the application programs have been written in a higher-level language, then it may be possible to preserve their code by modifying the compiler to reflect the architectural modifications, provided the strategies of the system design and implementation are still appropriate. A case in point is the design of machines which directly execute some higher-level language by providing primitives which closely correspond to statements in the language and data types available. The logic of software in that language is not affected, but its performance time can improve.

A New Independent Mode Of Operation

By adding to existing firmware new microprograms which create different instructions and a different architecture, it is possible to leave the original machine and its software unchanged and yet introduce new software which is matched to the new architecture. This approach extends the utility of the computer. A method of changing from one mode to another must be available. On the 360 series there is a means of running a 360/40 in its own mode or 1410 emulation mode. On small, inexpensive microprogrammable machines, such as the Interdata 3, the mode of operation can be changed by physically changing read-only memories. This is practical when the two modes are used at different times. The advent of writable control stores, such as those in the IBM 370 series, simplifies changing modes. It is possible to dynamically change the mode of operation of a machine and even operate processes in different modes concurrently under multiprogramming on such machines as the IBM 360/85 and the 370 series.

Adding Or Replacing Processors

The need to change operational computer-based systems has often occurred because they are overloaded and cannot cope with the amount of data or processing demanded of them. A notable example today is the Burroughs 3500 base computer. To combat this problem, the only feasible solution is often to supplement the available processor or to replace it. Consider the difficulties imposed when the machine in use is obsolete, and it is no longer possible or feasible to acquire additional computers of the same type or to provide additional capability on the existing machine. If the machine is not microprogrammable, then any of the previous approaches; i.e., modifying or adding microprograms, are not possible. The option of acquiring new hardware introduces a means for utilizing microprogramming to smooth the transition and to optimize the cost/performance ratio of the new processor hardware.

Replacement Of A Processor

It is highly probable that a replacement for a processor will be a microprogrammable machine which is larger and/or faster. The software can be moved to the new machine which can emulate the previous machine's instruction set in its microprogram. It may also be possible to extend the capabilities of the new machine in the ways described above so that existing software can gradually be phased over to a new architecture, or new applications can use new or additional capabilities. An example of this application of microprogrammed emulation is underway at SAMSO, where there are nine CDC 160A's which are overloaded, and there is too much software to reprogram for another machine. The decision was made to replace the CDC processors with META 4's which will emulate the CDC machine. The META 4 has a memory speed seven times faster than the CDC 160A. With emulation, the software should run up to six times faster than before.

Addition Of Processors

Another approach to adding capacity or decreasing processing time is to add new processors. A new processor may be compatible with an existing processor, although not the identical hardware, through emulation. If the new processor will serve a functionally different role, then microprogramming can be valuable in tailoring it to that role.

RESEARCH AND DEVELOPMENT TASKS

Data Management Systems

The management of data is what computers are for, according to users. Data management systems (DMS) contain the software to make it easier for users to describe the structure and relationships among data and how the data should be processed. Unlike a series of COBOL application programs which together make up all the data management functions but are individually independent, data management systems are composed of a set of programs which are highly interrelated, logically and physically. The DMS usually offers data management services consisting, at a minimum, of file creation, updating, retrieval, and report generation. The continuum of capabilities starts with tape-oriented, serial batch processing and moves conceptually, if not actually, to on-line, real-time data storage, retrieval, and reporting systems. Data management systems have become a focal point of concern because of their evident relevance to military applications. There is an increased awareness that present techniques cannot cope with the volumes of data already available or anticipated without becoming increasingly slow, unless the processing algorithms and the data

organization become more complex. This results in a longer time to develop the system and a degradation in reliability due to the additional complexity. Problems in the development of data management systems are discussed in detail by Glore [36].

The performance of current data management systems is slow compared to the requirements of many military applications. Serial, batch processing time increases rapidly as the size of the data base grows. Added complexity can increase the speed of some functions, such as retrieval, at the cost of others, such as updating. The size of programs, as well as the size of data, causes delays while limited amounts of high-speed memory must be filled and refilled. At the heart of these problems is the mismatch between the hardware and the demands of the data management processes. Faster processors and larger, faster memories are not economically feasible, nor would they resolve all of the difficulties in acquisition of data management systems. The architecture of conventional machines is inadequate for some of the kinds of processing which are needed. New hardware developments, such as content-addressable memories, are aimed at facilitating the rapid retrieval of data, but their application is restrained by the state of the technology. Changes in machine architecture can be made through microprogramming which will reduce the implementation and performance problems of data management systems. The Software Production task, described later, focuses on methods of improving programmer productivity and software reliability. The Multiprogramming and Multiprocessing task considers ways of improving performance by distributing processing among several processors. In this task, microprogramming is used to make the processes specific to data management systems more efficient. Microprogrammed changes will be made to speed up time-consuming parts of the execution time of current systems and to modify the way in which data is represented or processing is accomplished.

The following subtasks have been identified in the conduct of this task:

1. Select several data management functions and a simple data structure. Define the functions from the "outside in"; i.e., in terms of what a user would expect to happen.

2. Define algorithms for performing the functions. If there are several distinct methods of performing the same function, more than one algorithm might be specified.

3. The algorithms will be implemented in a high-level language, typical of the kind that would be used for a DMS. Most likely candidates are JOVIAL and PL/I. Any recommendations on programming languages and procedures resulting from the work on

Highly Reliable Programming under Project 5550 [37] will be incorporated in this activity. The machine code produced will be examined for gross inefficiencies due to the lack of optimization in the compiler, and some code may be replaced by machine code directly. The objective is to obtain a representative sample of machine code for some data management functions.

4. An analysis will be performed of the machine code to determine how parts of the code contribute to the execution time and the relationship between statements in the language and the amount of code that is generated. Instrumentation techniques are available for performance measurements of distribution of CPU and I/O time as well as how CPU time is spent among parts of the code. Both kinds of data will serve as a basis for selecting new machine features to be implemented in firmware which will conserve program space and speed up processing.

5. Variations in machine architecture will be devised to support the functions analyzed. Their effect will be derived by specifying the microprograms and estimating the execution time. The results will be reported and recommendations made for those microprogramming changes which are most beneficial to the particular functions analyzed and data management in general.

Security

The security of computer-based systems is concerned with controlling access to the information in the system, based on the level of classification of the user and of the data, and controlling processes exercised by users in order to access data or create data. Protection must be provided against accidental or overt destruction of processes and data as well as illegal access. The traditional method of providing security in data processing systems is to place the user and the computer in a secure environment and take electronic precautions to prevent any radiation from emanating out of that shielded environment. Such an approach is anachronistic in the face of current hardware and software technology permitting multiple users to time-share a computer and access it from remote locations. Three features of multi-user systems which make it difficult and complex to assure security of data have been described [38]:

1. "concurrent multiple users with different access rights operating remote from the shielded room;

2. multiple programs with different access rights co-resident in memory;

3. multiple files of different data sensitivities simultaneously accessible."

At MAC and at AF/AFACS, requirements have been expressed for a data processing capability which can handle mixed levels of classification on data, and both cleared and uncleared users, with some at terminals in uncleared areas.

The organization of most commercially available operating systems precludes their certification. The functions affecting security have not been sufficiently isolated, and the complex interrelationships of parts of the operating system prevent complete identification of modifications to provide security. There have been some notable attempts to handle the security problem with a combination of hardware and software [38, 39], but there are no time-sharing systems or multiprogramming systems in existence at this time which have been certified for operation in an open environment. The problems stem from the complexity of software, in the operating system as well as the data management facilities, which must be certified. At present, the problem of computer security is not well understood. There are no accepted criteria for certifying software design and implementation.

The solution to the security problem in information processing will probably involve a combination of hardware and software techniques. Firmware may also be of assistance to create a machine architecture which more carefully controls software and reduces the time penalty for monitoring activities. The Multics System [39] uses some hardware features such as addressing mechanisms and privileged instructions as expedients in controlling access. Firmware can provide these and other functions.

Subtasks of this task are the following:

1. Identify criteria for secure handling of data. The list will by no means be exhaustive, since this is a research subject in itself. Studies of the deficiencies in existing operating systems and partial solutions have identified some of the major kinds of protection that are needed.

2. Design architectural features to support these criteria and the means of achieving them in firmware for demonstration purposes.

3. Design the structure of an operating system which would use the new machine architecture. The objective is to minimize the amount of software that would require certification.

4. Implement the demonstration capability to illustrate and evaluate the principles. Heavy use will be made of an independent test team.

At a minimum, the results should increase understanding of the computer security problem. They can also contribute toward the establishment of criteria and methods for achieving them. An activity which relates to this research area is the work on Highly Reliable Programming in Project 5550 of the Advanced Development Program [37], which might be expected to make recommendations about methods of organizing and implementing procedures whose correctness might be proven. Certainly the parts of software which handle security are likely candidates.

Software Production

The amount of software which will be produced in support of computer-based Air Force applications over the next five to ten years is prodigious. One source [40] predicts that the Air Force will spend half its time on checkout and testing of software. With the current rate of programmer productivity estimated as from 125 to 500 instructions per man-month [41], it is clear that the production of software will be a major expenditure, surpassing the cost of hardware. The software which has already been produced has been notoriously unreliable and will continue to be so unless something is done to change the environment of the systems designers and programmers who produce the present and future computer-based systems.

A notable attempt to improve programmer productivity and software reliability was the development of high-order programming languages (HOL's). Such languages are intended to resemble the way in which a programmer conceptualizes his data and processing requirements. HOL's are labeled "machine-independent" to emphasize that HOL programs can be run on different machines with different architectures, but also to indicate that they are different from any machine architecture. The need for developing HOL's is a symptom of the fact that machine architectures and machine instructions are not suited to the humans that use them. The extent of this mismatch can be seen by the fact that the code produced by HOL compilers is not efficient of space or operating time, and a great deal of effort is being expended on devising means for compiler optimization of code. Despite this effort, it is difficult for HOL programs to be as efficient as machine language programs because the HOL shields the programmer from the important characteristics of the machine architecture such as word length and instruction set. This prevents the programmer from determining the consequences of alternative, logically equivalent, algorithms for performing a function. At the language level, all alternatives seem equally appealing, or the shortest number of statements may seem best. He must produce

the program and inspect or measure the results in order to determine efficiency. IBM supplies Programmer Guides to aid in the analysis of the effect of the machine and the compiler on the size and efficiency of the code produced.

Programmer productivity is estimated to be doubled by the use of HOL's. There is no evidence of the extent to which the software produced is less error-prone. Despite the inefficient code, their use is considered desirable by the Air Force for the implementation of large systems. There are two major ways in which microprogramming can be used to improve the effectiveness of HOL's in the software production process:

1. Modifications will be made to machine architecture where the features of a language which is suitable for a class of applications do not map well into the machine instruction set of a computer. The design of a machine language which more directly executes statements of a high-order language is feasible, and the design of compilers makes them amenable to this approach, as described for Airborne Systems in Section III. Some of this work will be carried out under the Data Management Systems task, described above, for that class of applications.

2. Even in the best of all possible worlds, implementation errors will occur, and predictions of performance will be wrong. Microprogramming offers the opportunity to extend the architecture of machines to allow the introduction of tools for monitoring the behavior of software, both to isolate errors and to analyze performance. In providing these services, conspicuously absent in most of the current computers, it becomes possible to achieve faster, more thorough testing of systems and to devote time to optimizing performance where it can be most effective.

Work in applying microprogramming to improving software production has been conducted at The MITRE Corporation (18, 23). A microprogram and software system, called the Venus Multiprogramming System, was developed specifically to facilitate the production of large software systems by teams of programmers. Microprogrammed features were provided to perform dynamic allocation of storage and to facilitate the use of many program modules in constructing a system. Machine primitives were added to permit systematic management of multiprogramming of cooperative, rather than independent, processes. This experience has shown the implications of the new architecture on the design and development of an operating system, a prime example of complex software. The hands-on use of the system has also suggested further uses of microprogramming, particularly to provide better aids to system testing and measurement. The subtasks below are a direct extension of the Venus work:

1. Analyze and report the benefits and deficiencies of the Venus firmware for software production.

2. Select and implement modifications and extensions to the Venus microprogram based on the analysis. Special attention will be given to support for debugging, and instrumentation aids, suitable for high-order language programming as well as machine language.

3. Apply the new architecture to a software implementation and evaluate the impact. In support of this activity, a compiler will be built to permit the use of the Venus System with a high-order language.

The relationship of this entire activity to the Data Management task has already been acknowledged. The microprogrammed changes arising from that task will be integrated with changes proposed by this task. It is suitable to use a data management application to evaluate both tasks.

Multiprogramming and Multiprocessing

Multiprogramming has developed as a way of more efficiently using the computer resources at hand, by trying to have the load equalized over the different parts of the system. Multiprocessing can provide a variety of services: faster execution time, when response time is critical; reliability for a system made up of a combination of similar components which can keep operating when some of the components fail; flexibility for making a system that can be tailored to cover a range of applications and computing power requirements. The Air Force and the Navy have expressed interest in the capabilities offered by multiprocessing in command and control (cf. post 1975 TACC requirements for a multiprocessor).

Both multiprogramming and multiprocessing, while often effective, introduce complexity to systems and create problems not found in simple systems. Some problems arise in defining parallel processes during system design and in designing the mechanisms for controlling asynchronous operations when concurrent processes cooperate in performing tasks. In addition, when multiprocessing is involved, the order in which computations should be performed is not necessarily the same as for a single processor system. The testing of such multi systems is particularly difficult because their asynchronous nature can create very subtle and unforeseen combination of events, their complexity makes it difficult to create or reconstruct test conditions at will, and it is impossible to test all the possible combinations of events.

One of the ways in which microprogramming can contribute to more effective use of asynchronous processes is to provide a machine architecture which permits systematic handling of concurrent activities. An approach is to provide machine operations to govern the sharing of data among the processes according to a set of principles which prevents conflicts in use of shared data. Operations can also be devised for synchronization among concurrent processes. The Venus Multiprogramming System, developed at MITRE, contains microprogrammed machine instructions which perform such functions in accordance with the work of Dijkstra [41], who defined operators for the purpose of synchronization as well as sharing data. A similar machine organization was proposed by Wirth [42], although microprogramming was not used to realize it. These techniques can be applied to the problem of cooperating processors as well as processes. Microprogramming can also facilitate the dynamic reconfiguration of the system when one component fails and modify scheduling for the reduced set of processors.

In developing larger multiprocessing systems, such as those which might be used for DMS applications, microprogramming offers the possibility of a partially specialized processor. In this case, any processor in the system would be able to perform any operation, but through the microprogramming different processors would be optimized for the performance of critical problems. The problem of scheduling the various special processors could also be handled on the microcode level.

Multiprogramming and multiprocessing have increased the importance of operating systems, and there is an obvious necessity for them to perform efficiently and reliably. Microprogramming can also be used for this purpose.

Subtasks which have been identified for the coming year.

1. Application of microprogramming techniques for the efficient and reliable operation of operating systems. More thought and research is needed to determine how to distribute the operating systems' functions between firmware and software. To support improved efficiency, monitoring must also be made of the system on both the software and firmware levels.

2. Application of microprogramming techniques for a system which can have a variable number of processors, from one to many. The area incorporates the complexities of designing such a variable system, dynamic reconfiguration and error detection. This is especially important in view of the TACC proposal.

3. Investigation of the appropriate types of association of processors to processes for different applications of multiprocessing.

4. Investigation of the applicability of semi-special processors for the use in multiprocessor systems, especially a DMS.

Microprogramming Techniques

To produce firmware is a programming job. Producing microprograms effectively is necessary in order to deliver the products of the research described above. In many ways, described earlier, microprogramming can be more difficult than software programming. While attention is being given to the design of microprograms and the tools for specifying, testing, and optimizing them, it is desirable to have a separate task to monitor this work. Where advances are made, they can be applied to the other research tasks. Deficiencies must also be recognized and solutions devised. New problems, not unlike those faced in software systems, will arise from the application of dynamically writable control stores.

CONCLUSIONS

The problems addressed by the proposed research exist today in the Air Force software systems being procured. None of the problems has an obvious or simple solution. Microprogramming is a tool which exists and can have an effect on the causes of the problems. Whether it is the best solution for a specific problem is difficult to assess, as shown in Section II. However, microprogramming is relatively inexpensive to apply for creating new computer architectures. Its flexibility permits changes and improvements to be made. For these reasons it is a compelling choice as a means of proving the acceptability of new machine features. It is then possible to determine whether these features can be realized in other ways, such as acquisition of new hardware, which are more cost-effective, if the microprogramming flexibility can be sacrificed in a particular situation.

REFERENCES

1. M. V. Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester University Computer Inaugural Conference, July 1951, Manchester, England, 16-18.

2. A. Opler, "Fourth Generation Software," Datamation, 13, 1, January 1967, 22-24.

3. R. F. Rosin, "Contemporary Concepts of Microprogramming and Emulation," Computing Surveys, 1, 4, December 1969, 197-212.

4. T. Berschback, Annotated Microprogramming Bibliography, M69-65, ESD-TR 70-204, AD-709-765, Contract F19(628)-68-C-0365, Bedford, Mass., July 1970.

5. S. S. Husson, "Microprogramming: Principles and Practices," Prentice-Hall, 1970.

6. L. Amdahl, "Architectural Questions of the Seventies," Datamation, 16, 1, January 1970, 66-68.

7. F. G. Withington, "Trends in MIS Technology," Datamation, 16, 2, February 1970, 108-119.

8. J. K. Wineke and M. Spiegel, "Generation IV: The Shape of Systems to Come," Computer Decisions, 2, 10, October 1970, 18-23.

9. R. Wolfe, "Multiple Minicomputers Go to Work for Large Timesharing Applications," Data Processing, 12, 9, (1970), 33-37.

10. A. W. Maholick and H. H. Schwarzell, "Integrated Microprogrammed Communication Control," Computer Design, 8, 11, November 1969, 127-131.

11. D. E. Waldecker, "Comparison of a Micro-programmed and a Non-microprogrammed Computer," Computer Design, 9, 6, June 1970, 73-78.

12. G. Hornbuckle and E. Ancona, "The LX-1 Microprocessor and Its Application to Real-Time Signal Proc.," IEEE Transactions on Computers, C-19, 8, August 1970, 710-720.

13. N. Bartow and R. McGuire, "System/360 Model 85 Microdiagnostics," AFIPS SJCC, 36, (1970), 191-197.

REFERENCES (Continued)

14. W. J. Patzer and G. C. Vandling, "Systems Implications of Microprogramming," Computer Design, 8, 12, December 1967, 62-67.

15. R. W. Cook and M. J. Flynn, "System Design of a Dynamic Microprocessor," IEEE Transactions on Computers, C-19, 3, March 1970, 213-222.

16. H. Barsamian, "Firmware Sort Processor with LSI Components," AFIPS SJCC, 36, (1970), 183-190.

17. C. W. Ramamoorthy and R. L. Kleir, "A Survey of Techniques for Optimizing Microprograms," Preprints of the 3rd Annual Workshop on Microprogramming, October 1970.

18. B. H. Liskov, The Venus Multiprogramming System - Year End Summary, The MITRE Corporation, MTR 2004, ESD-TR-70-408, Contract F19(628)-68-C-0365, Bedford, Mass., 31 August 1970.

19. H. A. Lichstein, "When Should You Emulate?," Datamation, 15, 11 November 1969, 205-210.

20. A. M. Johnson, "The Microdiagnostics for the IBM System/360 Model 30," Preprints of the 3rd Annual Workshop on Microprogramming, October 1970.

21. Computer Operations, Inc., Gemini Computer Systems Information Manual, Costa Mesa, Calif., (1970).

22. J. A. Howard and L. Pfeifer, "An ROM Bootstrap Loader for Small Computers," Computer Design, 9, 10, October 1970, 95-97.

23. B. J. Huberman, Principles of Operation of the Venus Microprogram, The MITRE Corporation, MTR 1843, ESD-TR-70-198, Contract F19(628)-68-C-0365, Bedford, Mass., 1 May 1970.

24. R. L. Brening, "External Control," Datamation, 16, 10, 1 September 1970, 48-55.

25. H. Burner, R. Million, O. Rechard, J. Sobolewski, A Programmable Data Concentralor for a Large Computing System, Washington State University, WSU-1969-1, 1 May 1969.

26. S. Matsushita, "A Microprogrammed Communication Control Unit, The TOSBAC DN-231," IFIP 68, Hardware Computer Systems, (1969), North Holland Publishing Co., 812-817.

27. W. C. McGee and H. E. Peterson, "Microprogram Control for the Experimental Sciences," Proceedings - AFIPS FJCC, 27, I, (1965), 77-91.

28. D. R. Doucette,"Performance Enhancement by Special Instructions on the System/360, Models 40 and 50," delivered at the Third Annual Workshop on Microprogramming, October 12-13, 1970.

29. J. F. Ruggiero and D. A. Coryell, "An Auxiliary Processing System for Array Calculations," IBM Systems Journal, 8, 2, (1969), 118-135.

30. E. H. Miller, "Reliability Aspects of the Variable Instruction Computer," IEEE Transactions on Electronic Computers, EC-16, 5, October 1967, 596-602.

31. "New Airborne Computer Concepts Evolve," Aviation Week and Space Technology, 22 June 1970, 213-219.

32. H. A. Weber, "Microprogrammed Implementation of Euler on IBM 360/30," Communications of the ACM, 10, 9, September 1967, 549-558.

33. R. S. Entner, "The Advanced Avionic Digital Computer System," Computer Design, 9, 9, September 1970, 73-76.

34. The Blue Ribbon Defense Panel, Report to The President and The Secretary of Defense on the Department of Defense, 1 July 1970, 153.

35. G. E. Hoernes and L. Hellerman, "An Experimental 360/40 for Time Sharing," Datamation, April 1968.

36. J. Glore, Major Problems of Generalized Data Management System Development, M70-56, The MITRE Corporation, February 1970.

37. Development Plan RCS: DD-DREE(AR)637, Development Plan, Advanced Development Program, Data Processing Hardware and Software Technology, Project 5550, Air Force Systems Command, November 1970.

38. C. Weissman, "Security Controls in the ADEPT-50 Time-Sharing System," AFIPS Fall Joint Computer Conference, (1969), 119-133.

39. E. L. Glaser, "A Brief Description of Privacy Measures in the Multics Operating System," AFIPS Spring Joint Computer Conference, (1967), 303-304.

REFERENCES (Concluded)

40. B. W. Boehm, The RAND Corporation, Some Information Processing Implications of Air Force Space Missions: 1970-1980, RM-6213-PR, Santa Monica, Calif., January 1970.

41. E. Dijkstra, "The Structure of the 'THE' - Multiprogramming System," Communications of the ACM, 11, 5, May 1968.

42. N. Wirth, "On Multiprogramming, Machine Coding, and Computer Organization," Communications of the ACM, 12, 9, September 1969, 489-498.

43. J. D. Aron, "Estimating Resources for Large Programming Systems," Software Engineering Techniques, NATO Science Committee, Brussels, Belgium, (1970), 68-79.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
The MITRE Corporation Bedford, Massachusetts	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

THE APPLICATION OF MICROPROGRAMMING TECHNOLOGY

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

Judith A. Clapp

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
MAY 1971	45	43

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F19(628)-71-C-0002	ESD-TR-71-105
b. PROJECT NO. 6710	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	MTR-2050

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts

13. ABSTRACT

This report surveys promising applications of microprogramming. Emphasis is on the value of microprogramming as a tool which permits computer users to modify the architecture of a general-purpose machine to better match a particular set of requirements. Factors are discussed which affect the choice of microprogramming over hardware and software in the design and implementation of computer-based systems. Actual and potential examples of its application are given to illustrate its relevance to the solution of implementation and performance problems arising in typical Air Force systems. Finally, research and development tasks are proposed which lead to the realization of the benefits of this technology in operational command control and communications systems. Methods are described for integrating the results into existing and future systems in the next several years.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
AIRBORNE COMPUTER SYSTEMS						
COMPUTER PROGRAMMING						
COMPUTER SYSTEMS PROGRAMS						
COMPUTER PROGRAMS						
COMMUNICATIONS PROCESS						
INPUT-OUTPUT ROUTINES						
MICROPROGRAMMING						
PROGRAMMING TECHNIQUES						
PROGRAMS (COMPUTERS)						
REAL TIME OPERATIONS						
SOFTWARE (COMPUTERS)						